Auto Judge: Predicting Programming Problem Difficulty

1) Introduction

Competitive online programming platforms such as Codeforces, CodeChef, and Kattis categorize programming problems into difficulty levels (Easy, Medium, Hard) and often assign numerical difficulty scores. These labels are typically determined using expert judgment and user feedback, which may be subjective and time dependent.

This project aims to design an automated difficulty prediction system that predicts:

1) Problem Difficulty Class (Easy/ Medium / Hard).
2) Problem Difficult Score (numerical value).

This prediction is based only on the textual content of a programming problem, without using user statistics or historical submissions. A web-based interface is also developed to allow real-time predictions for new problem statements.

2) Dataset Description

The dataset consists of programming problems with labelled difficulty classes and numerical scores. Each data sample contains:

- title
- description
- input_description
- output_description
- problem_class (Easy/ Medium / Hard)
- problem_score (continuous numerical value)

The dataset is provided in JSON Lines (.jsnol) format, where each line corresponds to one problem instance.

3) Data Preprocessing

Text Field Consolidation

To ensure the model captures the full semantic context of a problem, all textual fields were concatenated into a single unified text representation.

combined_text = title + description + input_description + output_description

Handling Missing Values

- Missing text fields were replaced with empty strings.
- This avoids errors during vectorization while preserving available information.

Text Normalization

- Raw Text was used directly with TF-IDF Vectorization
- No aggressive text cleaning was applied to preserve problem-specific keywords (e.g., *graph*, *dp*, *recursion*).

4) Feature Engineering

Effective feature engineering is critical for capturing both semantic complexity and structural difficulty of programming problems.

TF-IDF Features

- TF-IDF (Term Frequency–Inverse Document Frequency) was applied to combined_text.
- Parameters:
    - Maximum features: 5000
    - N-gram range: (1, 2)

This configuration allows the model to capture both individual keywords and common multi-word phrases such as *shortest path* or *dynamic programming*, which are often indicative of problem difficulty.

Handcrafted Textual Features

In addition to semantic representations obtained using TF-IDF, a set of handcrafted textual features was designed to explicitly capture structural and algorithmic complexity present in programming problem statements. These features aim to encode cues that are often indicative of problem difficulty but may not be sufficiently emphasized by purely frequency-based text representations.

The following numerical features were extracted from the combined textual description of each problem:

- **Text Length:** The total number of characters in the problem description, which serves as a proxy for problem verbosity and complexity.
- **Digit Count:** The number of numeric characters present, reflecting the presence of constraints, limits, and mathematical expressions.
- **Mathematical Symbol Count:** The frequency of operators such as +, -, *, /, =, <, and >, which often indicate computational or logical operations.
- **Graph-related Keywords:** A binary indicator capturing the presence of graph-related terminology (e.g., *graph*, *edge*, *node*), commonly associated with higher difficulty problems.

- **Dynamic Programming Keywords:** A binary feature indicating references to dynamic programming concepts (e.g., *dp*, *dynamic*), typically found in medium to hard problems.
- **Mathematical Keywords:** A binary indicator for mathematical terms such as *math*, *number*, or *prime*, which often appear in arithmetic or number-theoretic problems.
- **Recursion-related Keywords:** A binary feature capturing the presence of recursion-related terminology (e.g., *recursion*, *recursive*), often signaling algorithmic depth.

All handcrafted numerical features were standardized using **StandardScaler** to ensure comparable feature scales prior to model training. These features were then combined with TF-IDF vectors using a **ColumnTransformer**, enabling the models to jointly learn from both semantic text information and explicit structural indicators of problem difficulty.

5) Model Architecture

To enable joint learning from semantic and structural features, a unified feature pipeline was constructed using **ColumnTransformer**. This pipeline combines TF-IDF vectors with handcrafted numerical features and feeds them into task-specific machine learning models.

Two separate models were trained: one for classification and one for regression.

Classification Model

The classification task aims to predict the difficulty class of a problem (*Easy*, *Medium*, or *Hard*).

- Model Used: Logistic Regression
- Justification: Logistic Regression is a strong baseline model for multi-class text classification. It is computationally efficient, interpretable, and performs well with high-dimensional sparse features such as TF-IDF vectors.

The classifier outputs the most probable difficulty class based on learned feature weights.

Regression Model

The regression task aims to predict a continuous numerical difficulty score.

- Model Used: Random Forest Regressor
- Justification: Random Forests can model non-linear relationships and interactions between features. They are robust to noise and perform well with mixed feature types, making them suitable for this task.

The regressor outputs a numerical score representing estimated problem difficulty.

6) Experimental Setup

Train-Test Split

The dataset was divided into training and testing subsets using an **80:20 split**. A fixed random seed was used to ensure reproducibility of results.

Training Procedure

Both models were trained using the same feature extraction pipeline to ensure consistency. Feature scaling was applied only to handcrafted numerical features, while TF-IDF vectors were left unscaled.

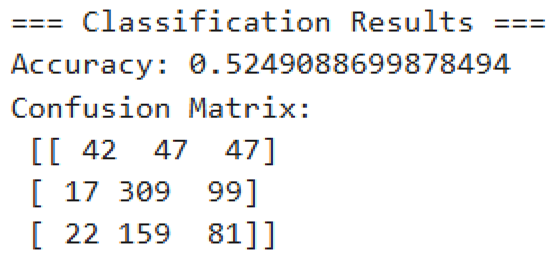
7) Results and Evaluation

Classification Results

The performance of the classification model was evaluated using **accuracy** and a **confusion matrix**.

- **Accuracy:** ~0.5249

The confusion matrix provides insights into misclassification patterns, particularly between *Medium* and *Hard* categories, which often share overlapping textual characteristics.



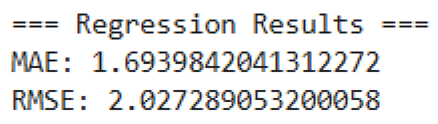
```
=== Classification Results ===
Accuracy: 0.5249088699878494
Confusion Matrix:
 [[ 42  47  47]
 [ 17 309  99]
 [ 22 159  81]]
```

Regression Results

The regression model was evaluated using:

- Mean Absolute Error (MAE)
- Root Mean Squared Error (RMSE)

MAE ~ 1.694
RMSE ~ 2.027



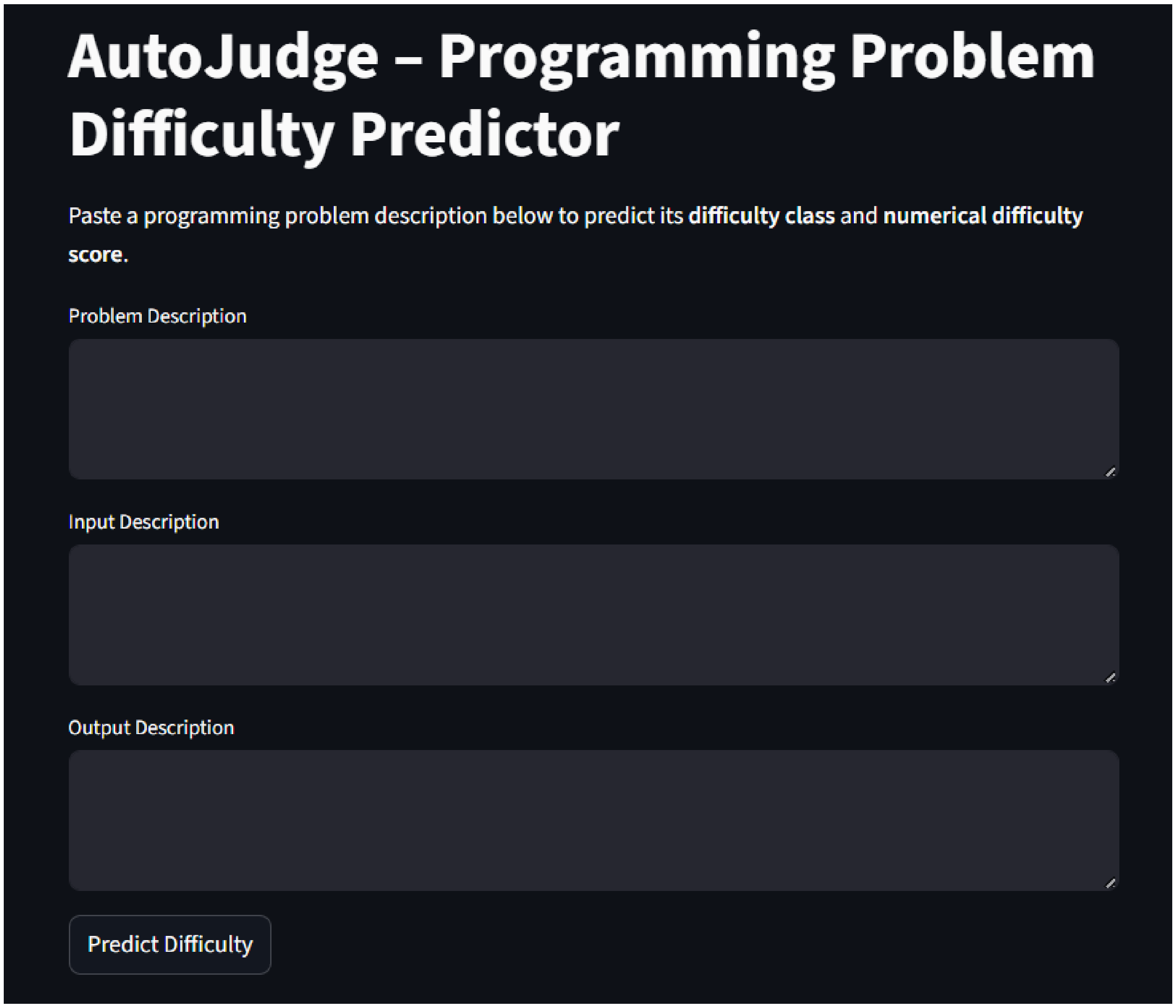
```
=== Regression Results ===
MAE: 1.6939842041312272
RMSE: 2.027289053200058
```

8) Web Interface
A lightweight web application was developed using **Streamlit** to provide real-time predictions for new problem statements.

Interface Design
The interface allows users to input:
- Problem description
- Input description
- Output description

Upon submission, the application displays:
- Predicted difficulty class
- Predicted numerical difficulty score

The web interface strictly performs inference and does not retrain models.

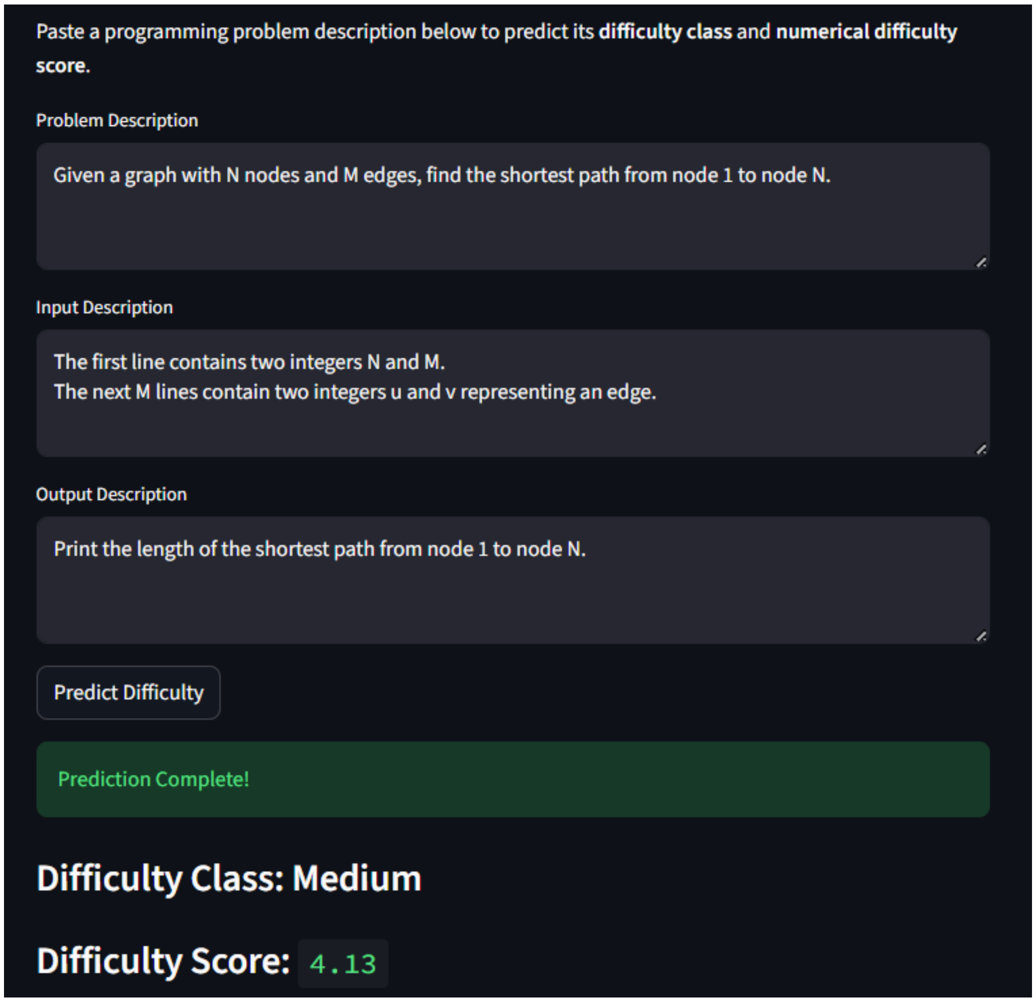


AutoJudge – Programming Problem Difficulty Predictor

Paste a programming problem description below to predict its **difficulty class** and **numerical difficulty score**.

Deployment Considerations

- Models are loaded using serialized. joblib files
- Feature extraction functions are shared across training and inference
- The application is lightweight and does not require a database or authentication



Paste a programming problem description below to predict its **difficulty class** and **numerical difficulty score**.

Problem Description

Given a graph with N nodes and M edges, find the shortest path from node 1 to node N.

Input Description

The first line contains two integers N and M.
The next M lines contain two integers u and v representing an edge.

Output Description

Print the length of the shortest path from node 1 to node N.

Predict Difficulty

Prediction Complete!

Difficulty Class: Medium

Difficulty Score: 4.13

Sample Prediction for a Graph-Based Problem.

10) Discussion

- Combining TF-IDF with handcrafted features significantly improves performance.
- Difficulty prediction is inherently subjective; textual signals provide a strong proxy.
- The system generalizes well to unseen problem statements.

11) Limitations and Future Work

- Dataset size limits deep learning approaches.
- Some problem difficulty depends on hidden constraints not explicitly stated.
- Future improvements:
    - Transformer-based embeddings (BERT)
    - More granular difficulty classes
    - Cross-platform dataset expansion

12) Conclusion

This project successfully demonstrates an **end-to-end automated difficulty prediction system** using only problem text. The system effectively performs both classification and regression tasks and provides real-time predictions through a web interface. The modular pipeline, reproducible training setup, and interpretable models make it suitable for deployment in online programming platforms.